

Comparative Evaluation of Symmetric SVD Algorithms for Real-time Face and Eye Tracking

Tapan Pradhan, Aurobinda Routray, and
Bibek Kabi

Department of Electrical Engineering, IIT Kharagpur,
Kharagpur, India - 721302
{tapanpradhan78,aurobinda.routray,bibek.kabi}@gmail.com

Abstract. Computation of singular value decomposition (SVD) has been a topic of concern by many numerical linear algebra researchers. Fast SVD has been a very effective tool in computer vision in a number of aspects, such as: face recognition, eye tracking etc. At the present state of the art fast and fixed-point power efficient SVD algorithm needs to be developed for real-time embedded computing. The work in this paper is the genesis of an attempt to build an on-board real-time face and eye tracking system for human drivers to detect loss of attention due to drowsiness or fatigue. A major function of this on-board system is quick customization. This is carried out when a new driver comes in. The face and eye images are recorded while instructing the driver for making specific poses. The eigen faces and eigen eyes are generated at several resolution levels and stored in the on-board computer. The discriminating eigen space of face and eyes are determined and stored in the on-board flash memory for detection and tracking of face and eyes and classification of eyes (open or closed) as well. Therefore, fast SVD of image covariance matrix at various levels of resolution needs to be carried out to generate the eigen database. As a preliminary step, we review the existing symmetric SVD algorithms and evaluate their feasibility for such an application. In this article, we compare the performance of (1) Jacobi's, (2) Hestenes', (3) Golub-Kahan, (4) Tridiagonalization and Symmetric QR iteration and (5) Tridiagonalization and Divide and Conquer algorithms. A case study has been demonstrated as an example.

Keywords: Fast SVD, Eigen Space, Face and Eye Detection

1 Introduction

SVD is a powerful tool in digital signal and image processing. It states that a matrix can be decomposed as follows,

$$A = U \Sigma V^T \quad (1)$$

Where $A_{m \times n}$ is a dense matrix, $U_{m \times m}$ and $V_{n \times n}$ are orthogonal (or unitary) matrices and their columns are called left and right singular vectors respectively.

Σ is a diagonal matrix and contains all singular values along its diagonal in a non-increasing order. For a symmetric matrix $m = n$ and U and V span the same vector space. Hence computation of either U or V is sufficient. For any dense symmetric matrix $A_{n \times n}$, Eigen Value Decomposition (EVD) is defined as follows

$$A = X \Lambda X^T \quad (2)$$

Where X is the matrix of eigenvectors and Λ is the diagonal matrix containing eigenvalues along its diagonal. For symmetric matrices, eigenvalue decompositions and singular value decompositions are closely related as follows [5]: Suppose that A is a symmetric matrix, with eigenvalues λ_i and orthonormal eigen vectors u_i so that $A = U \Lambda U^T$ is an eigenvalue decomposition of A , with $\Lambda = \text{diag}[\lambda_1 \lambda_2 \dots \lambda_n]$, $U = [u_1 u_2 \dots u_n]$ and $U U^T = I$. Then an SVD of symmetric matrix A is, $A = U \Sigma V^T$, where diagonal elements of Σ i.e. $\sigma_i = \text{abs}(\lambda_i)$ and $v_i = \text{sign}(\sigma_i).u_i$ where $\text{sign}(0) = 1$. For symmetric positive definite matrices eigenvalue decomposition (EVD) and SVD leads to the same decomposition. Hence we will use eigenvalues/eigenvectors and singular values/singular vectors interchangeably.

However, SVD has been an offline tool for digital signal and image processing applications for decades because of the computation complexity and memory requirement. Due to the increased resources of some of the recently introduced workstations, there are attempts to develop faster versions of SVD algorithm for real-time signal and image processing applications. The implementation of SVD in embedded platforms like DSPs, ARM and FPGAs is necessary for facilitating efficient real-time image processing. Most of these platforms either have fixed-point processors or CLBs (Configurable Logic Blocks) to make the system cheaper and power efficient. Hence *fast and fixed-point SVD* algorithms are to be developed for such applications. The purpose of this work is to evaluate the existing SVD algorithms for their suitability on embedded platforms.

In pattern recognition, eigenspace based method has been proposed for face tracking or face recognition [1]-[4]. To find the eigenspace, SVD (or eigenvalue decomposition) is used. There are several algorithms for SVD as stated in literature [5]-[7]. Jacobi's algorithm is known to be the oldest and slowest algorithm [5]-[8]. For symmetric matrices, though Jacobi's algorithm generates accurate singular values and singular vectors, the time of execution increases with the dimension of the matrices and is only suitable as an offline tool. Two-sided and one-sided variants for Jacobi's algorithm are stated in literature. Hestenes' algorithm is a variant of one-sided Jacobi's algorithm and is discussed in [9]-[10]. Being a one-sided version the time of computation is lesser than two-sided Jacobi's algorithm. However, as the iteration is applied on the whole process this algorithm is also not suitable for online applications. Golub and Kahan proposed a two-step algorithm [6],[11]-[12] for computation of SVD. In the first step, a dense symmetric matrix is converted to a bidiagonal matrix, which is eventually

converted to a diagonal matrix using implicit QR iteration in the second step. Because the second phase of Golub-Kahan algorithm is iterative in nature, it is much faster than Jacobi's or Hestenes' algorithm. A similar two step algorithm has been proposed for SVD, where a dense symmetric matrix is reduced to tridiagonal matrix and then an implicit symmetric QR iteration is applied to reduce the symmetric tridiagonal matrix into a diagonal matrix. This algorithm is found to be faster and competitive with Golub-Kahan algorithm when a combination of QR and QL algorithm is used [5]-[7]. Still these algorithms could not satisfy real-time constraints as required by the signal and image processing platforms. Hence a Divide and Conquer algorithm was proposed by J.J.M. Cuppen based on a rank-one modification by Bunch, Nielsen and Sorensen. This is the fastest algorithm till date when a complete eigensystem of a symmetric tridiagonal matrix is required [5]. A variant of the said algorithm by Gu and Eisenstat has been implemented in LAPACK routine for matrices with dimension larger than 25 [13]-[14].

Faster performance is achieved when floating-point SVD is converted to fixed-point format and implemented in fixed-point platform [22]. Fast and Fixed-point SVD algorithm is also useful for reducing silicon area and power consumption in embedded platforms [24]-[25]. For Digital Signal Processing applications, attempts have been made to implement SVD algorithm using multiprocessor arrays [26] and CORDIC (COordinate Rotation DIgital Computer) based reconfigurable systems [27]-[28].

2 Existing Algorithms for SVD of Symmetric Matrices

In this section, we analyse different SVD algorithms for symmetric matrices along with their complexity, advantages and disadvantages.

1. Jacobi's Algorithm [8]
2. Hestenes' Algorithm [10]
3. Golub-Kahan Algorithm [11]-[12]
4. Tridiagonalization and Symmetric QR iteration [5]-[7]
5. Tridiagonalization and Divide and Conquer Algorithm [5]
6. Bisection and Inverse Iteration [5]

2.1 Jacobi's Algorithm

Jacobi's algorithm is the oldest and slowest available method for computing SVD by implicitly applying iteration on a dense symmetric matrix A . The method is more or less similar to the method for eigenvalue decomposition of symmetric matrices. Jacobi's method computes SVD of a symmetric matrix A with higher relative accuracy when A can be written in the form $A=DX$, where D is diagonal and X is well conditioned [5]. Thus

$$J^T A J = \Sigma \quad (3)$$

Table 1. Computational complexities of SVD algorithms

Algorithms	Time Complexity
Jacobi's Algorithm	$O(n^3)$
Hestenes Method	$O(n^3)$
Golub-Kahan Algorithm (Bidiagonalization + Implicit QR Iteration)	$(\Sigma[(\frac{8}{3}n^3 + O(n^2))] \text{ and } U, V[4n^3 + O(n^3)])$
Tridiagonalization + Symmetric QR Iteration	$8\frac{2}{3}n^3 + O(n^2)$
Tridiagonalization + Divide and Conquer Method	$\frac{8}{3}n^3 + O(n^2)$

In each step we compute a Jacobi rotation with J and update A to $J^T A J$, where J is chosen in such a way that two off-diagonal entries of a 2×2 matrix of A are set to zero in $J^T A J$. This is called two-sided or classical Jacobi method. Again, this can be achieved by forming $G = A^T A$ and performing iteration over G instead of A . The eigenvalues of G are the square of the singular values of A . Since $J^T G J = J^T A^T A J = (A J)^T (A J)$, we can obtain Σ by merely computing $A J$. This is termed as one-sided Jacobi rotation. Though Jacobi's algorithm for calculating SVD is the slowest among all presently available algorithms, it produces relatively accurate singular values and singular vectors. Jacobi's algorithm can be implemented in parallel as the individual steps are not interdependent. Parallel Jacobi's algorithm is discussed in [15].

Table 2. Two-sided Jacobi's algorithm for SVD

$Q = I$ (I = identity matrix)	...
repeat	s=c.t
for $j = 1 : (n - 1)$	$J(j, j) = c, J(k, k) = c, J(j, k) = s, J(k, j) = -s$
for $k = (j + 1) : n$	$A = J^T A J$
if $ A(j, k) $ is not too small	$Q = Q.J$
$\xi = \frac{(A(k, k) - A(j, j))}{2A(j, k)}$	endif
if $\xi = 0$	endfor
$t = \frac{1}{(\xi + \sqrt{1 + \xi^2})}$	endfor
else	$\sigma_k = A(k, k) $
$t = \frac{\text{sign}(\xi)}{(\xi + \sqrt{1 + \xi^2})}$	$U = Q$
endif	$v_k = \text{sign}(A(k, k)).u_k$
$c = \frac{1}{\sqrt{1 + t^2}}$	$V = [v_1 v_2 \dots v_n]$
contd ...	

2.2 Hestenes' Algorithm

Hestenes' algorithm is based on one-sided Jacobi's algorithm. A symmetric matrix of size $n \times n$ is used to generate an orthogonal rotation matrix V so

that the transformed matrix $A' = AV = W$ has orthogonal columns. Now if we normalize each non-null column of matrix W to unity, we get the relation $W = AV = U\Sigma = U \cdot \text{diag}(\sigma_1, \sigma_2, \dots, \sigma_n)$, $A = U\Sigma V^T$, $V = \prod_{i=1}^n J_i$ and singular values of A are $\sigma_i = \|a'_i\|_2$, where $A' = [a'_1 a'_2 \dots a'_n]$.

Table 3. Hestenes' algorithm for SVD

$V = I$...
repeat until convergence	$c = \frac{1}{\sqrt{1+t^2}}, s = t.c$
for $p = 1 : (n - 1)$	end
for $q = (p + 1) : n$	$J=I$
$\alpha = A(:, p)^T A(:, p), \beta = A(:, q)^T A(:, q)$	$J(p, p) = c, J(q, q) = c, J(p, q) = s, J(q, p) = -s$
$\gamma = A(:, p)^T A(:, q)$	$A = A.J, V = V.J$
if $\gamma = 0$	endfor
$c = 1; s = 0$	endfor
else	for $k = 1 : n$
$\zeta = \frac{(\beta - \alpha)}{2\gamma}$	$\sigma_i = \ A(:, k)\ $
$t = \frac{\text{sign}(\zeta)}{(\zeta + \sqrt{1 + \zeta^2})}$	end
contd ...	$U = AV \Sigma^{-1}$

2.3 Golub-Kahan Algorithm

This algorithm can be segmented in two phases,

- In the first phase a dense symmetric matrix is converted to a bi-diagonal matrix by orthogonal similarity transformation.
- This bi-diagonal matrix is then converted to diagonal matrix using Implicit QR iteration.

The standard algorithm may compute singular values with poor relative accuracy. However, with modified algorithm by Demmel-Kahan smaller singular values may be computed with high relative accuracy [12]. This algorithm has two steps as shown in equation (4).

$$\begin{pmatrix} * & * & * & * \\ * & * & * & * \\ * & * & * & * \\ * & * & * & * \end{pmatrix} \xrightarrow[\text{STEP I}]{U_1 A V_1^T = B} \begin{pmatrix} * & * & 0 & 0 \\ 0 & * & * & 0 \\ 0 & 0 & * & * \\ 0 & 0 & 0 & * \end{pmatrix} \xrightarrow[\text{STEP II}]{U_2 B V_2^T = \Sigma} \begin{pmatrix} * & 0 & 0 & 0 \\ 0 & * & 0 & 0 \\ 0 & 0 & * & 0 \\ 0 & 0 & 0 & * \end{pmatrix} \quad (4)$$

Hence, $U_2^T B V_2 = U_2^T U_1^T A V_1 V_2 = (U_1 U_2)^T A V_1 V_2 = U^T A V = \Sigma$ and Σ is the diagonal matrix containing singular values of the symmetric matrix A . In bi-diagonalisation process we have used householder method to make the elements

Table 4. Golub-Kahan algorithm for SVD

<pre> function [u, σ]=houszero(x) m = max(x_i), i = 1, 2, ..., n u_i = $\frac{x_i}{m}$, i = 1, 2, ..., n sign(0) = 1 σ = sign(u₁)√u₁² + u₂² + ... + u_n² u₁ = u₁ + σ σ = -m.σ end houszero </pre>	<pre> function [c, s, r]=rot(f,g) if f = 0 then c = 0; s = 1; r = g elseif (f > g) then t = $\frac{g}{f}$; tt = √(1 + t²); c = $\frac{1}{tt}$; s = t.c; r = tt.f else t = $\frac{f}{g}$; tt = √(1 + t²); s = $\frac{1}{tt}$; c = t.s; r = tt.g endif end rot </pre>
<p>Golub-Kahan algorithm for SVD</p> <pre> U₁ = I, V₁ = I for k = 1 : (n - 1) [u, σ] = houszero(A(k : m, k)) H1 = I - 2$\frac{uu^T}{u^T u}$ P₁ = I P₁(k : m, k : n) = H1 A(k : m, k : n) = H1.A(k : m, k : n) U₁ = U₁.P₁ if k ≤ (n - 2) [v, σ] = houszero(A(k, k + 1 : n)^T) H2 = I - 2$\frac{vv^T}{v^T v}$ P₂ = I P₂(k : m - 1, k : n - 1) = H2 A(k : m, k + 1 : n) = A(k : m, k + 1 : n).H2 V₁ = V₁.P₂ endif endfor contd ... </pre>	<pre> ... repeat for i = 1 : (n - 1) U₂ = I [c, s, r] = rot(A(i, i), A(i, i + 1)) Q = I Q(i : i + 1, i : i + 1) = $\begin{pmatrix} c & s \\ -s & c \end{pmatrix}$ A = A.Q^T V₂ = V₂.Q^T [c, s, r] = rot(A(i, i), A(i + 1, i)) Q = I Q(i : i + 1, i : i + 1) = $\begin{pmatrix} c & s \\ -s & c \end{pmatrix}$ A = Q.A U₂ = U₂.Q endfor Σ = abs(A) U^T = U₁.U₂ V = V₁.V₂ </pre>

of a column or row zero. Another optimized variant of this algorithm is also available and is known as Golub-Kahan-Chan algorithm.

2.4 Tridiagonalization and Symmetric QR iteration

In this method, the dense symmetric matrix is first converted to symmetric tridiagonal matrix in finite number of steps and the tridiagonal matrix is eventually converted to diagonal form by symmetric QR iteration. When a combination of QL and QR algorithm instead of QR is used, a stable variant is obtained. The steps of this algorithm is stated in equation (5).

Method of tridiagonalization

Householder reflection can be used for tridiagonalization. In this method, one row or column is picked up, and householder matrix is reconstructed to make all the elements of the row or column zero except the first one [6].

Hessenberg reduction [[7]] is a process by which a dense matrix is converted to an upper or lower Hessenberg matrix by orthogonal similarity transformation and thus does not change the eigenvalues or singular values. For a dense symmetric matrix, this Hessenberg reduction produces an upper as well as a lower Hessenberg matrix or a symmetric tridiagonal matrix which is used in symmetric QR iteration process to produce a diagonal matrix. In this process, the eigenvalues or singular values remain same as that of the original dense symmetric matrix. Householder or Givens method may be applied for Hessenberg reduction. We have used Givens method to produce symmetric tridiagonal matrix.

Lanczos method is useful to transform a symmetric dense matrix to a symmetric tridiagonal matrix. This method suffers from loss of orthogonality among Lanczos vectors with increased number of iterations. For this reason reorthogonalization is required to obtain proper orthogonal vectors.

Symmetric QL and QR iteration :

This is an iterative process and has a complexity of $O(n^2)$. After reducing the dense symmetric matrix into its tridiagonal form by orthogonal similarity transformation, symmetric QL or QR iteration is performed depending on which of the first and last diagonal elements of the symmetric tridiagonal matrix is larger. If the first diagonal entry is larger than the last one, QR is called upon to perform, else QL is called.

$$\begin{pmatrix} * & * & * & * \\ * & * & * & * \\ * & * & * & * \\ * & * & * & * \end{pmatrix} \xrightarrow[\text{STEP I}]{U_1 A V_1^T = T} \begin{pmatrix} * & * & 0 & 0 \\ * & * & * & 0 \\ 0 & * & * & * \\ 0 & 0 & * & * \end{pmatrix} \xrightarrow[\text{STEP II}]{U_2 T V_2^T = \Sigma} \begin{pmatrix} * & 0 & 0 & 0 \\ 0 & * & 0 & 0 \\ 0 & 0 & * & 0 \\ 0 & 0 & 0 & * \end{pmatrix} \quad (5)$$

Table 5. Tridiagonalization and Symmetric QR Iteration

function $[c, s] = \mathbf{givens}(a, b)$	$P_1(k+1 : n, k+1 : n) = H$
if $b = 0$	$A = P_1 A P_1$
$c = 1; s = 0$	$U_1 = U_1 P_1$
else	end
if $ b > a $	repeat
$\tau = -\frac{a}{b}$	$d = \frac{(t_{n-1, n-1} - t_{n, n})}{2}$
$s = \frac{1}{\sqrt{1+\tau^2}}; c = s \cdot \tau$	$\mu = t_{n, n} - \frac{t_{n, n-1}^2}{(d + \text{sign}(d) \sqrt{d^2 + t_{n, n-1}^2})}$
else	$x = t_{11} - \mu; z = t_{21}$
$\tau = -\frac{a}{b}$	for $k = 1 : (n-1)$
$c = \frac{1}{\sqrt{1+\tau^2}}; s = \frac{\tau}{\tau}$	$[c, s] = \mathbf{givens}(x, z)$
endif	$T = (G_k^T) T G_k$, where $G_k = G(k, k+1, \theta)$
.....	$U_1 = U_1 G_k$
$U_1 = I$	if $k < (n-1)$
for $k = 1 : (n-2)$	$x = t_{k+1, k}$
$[u, \sigma] = \mathbf{houszero}(x)$	$z = t_{k+2, k}$
$H = I_1 - 2 \frac{(uu^T)}{(u^T u)}$	endif
$P_1 = I$	endfor
contd..	$\Sigma = \text{abs}(T), U = U_1$

2.5 Tridiagonalization and Divide and Conquer Algorithm

The method was first proposed by J.J.M. Cuppen [14] based on a rank-one modification by Bunch, Nielsen and Sorensen [13]. However, the algorithm became popular after a stable variant for finding singular vectors or eigenvectors was found out in 1990 by Gu and Eisenstat [17]. This algorithm is the fastest SVD algorithm available till date. A dense symmetric matrix is first converted to symmetric tridiagonal matrix [5], [13]-[14]. Then the symmetric tridiagonal matrix is divided into two parts by rank one update and again each of the smaller matrices is divided till sufficiently smaller (matrix dimension = 25) matrices are formed. Then QR and QL iteration may be applied to find the SVD of the smaller matrices and using rank one update smaller solutions are combined together to form the complete SVD of the symmetric tridiagonal matrix. With a combination of previously stated steps, a complete eigensystem of a dense symmetric matrix can be found out.

Two major parts for finding the eigensystem of a symmetric tridiagonal matrix are *divide* and *conquer*. It works by breaking down a problem into two or more subproblems of the same type until the subproblem becomes simple enough to be solved directly. The solutions to the subproblems are then combined to generate a solution to the original problem. The most significant part is the solution of secular equation. The solution of the secular equation involves function approximation for finding the desired roots. For matrices with dimension greater

than 25 this is the fastest method for finding the complete eigensystem of a symmetric tridiagonal matrix till date [5].

Structure of the Algorithm [5] This includes dividing the symmetric tridiagonal matrix into two parts by removing the subdiagonal entry by rank one modification. Now with the known eigensystem of the two new symmetric tridiagonal matrices the secular equation is constructed. Solution of secular equation gives the eigenvalues of the original matrix from which the eigen vectors can also be computed. A dense symmetric matrix is converted to a symmetric tridiagonal matrix and then the divide and conquer algorithm is applied as [17].

1. Step 1 - Dividing the symmetric tridiagonal matrix into smaller matrices by rank one modification.
2. Step 2 - Sorting eigenvalues and eigenvectors obtained from smaller matrices in an increasing order using permutation matrix.
3. Step 3 - Deflation (when updation of eigenvalues and eigen vectors is not required) due to smaller coefficient or smaller components in the vector used for rank one modification and also for repeated eigenvalues are considered.
4. Step 4 - Formation and solution of secular equation using non-deflated eigenvalues.
5. Step 5 - Combining the solution obtained from secular equation solver and deflated eigenvalues to obtain the complete eigensystem of the symmetric tridiagonal matrix.

Forming the symmetric tridiagonal matrix from a dense symmetric matrix

$$U^T A U = T \quad (6)$$

A - Dense symmetric matrix

T - Symmetric tridiagonal matrix

U - Eigenvector matrix obtained from tridiagonalization process

Divide and Conquer Algorithm

Step 1 Rank one modification of the symmetric tridiagonal matrix

$$T = \begin{pmatrix} T_1 & 0 \\ 0 & T_2 \end{pmatrix} + b_m v v^T = \begin{pmatrix} Q_1 D_1 Q_1^T & 0 \\ 0 & Q_2 D_2 Q_2^T \end{pmatrix} + b_m v v^T \quad (7)$$

$$T = \begin{pmatrix} Q_1 & 0 \\ 0 & Q_2 \end{pmatrix} \left(\begin{pmatrix} D_1 & 0 \\ 0 & D_2 \end{pmatrix} + b_m u u^T \right) \begin{pmatrix} Q_1^T & 0 \\ 0 & Q_2^T \end{pmatrix} \quad (8)$$

Let, $D = \begin{pmatrix} D_1 & 0 \\ 0 & D_2 \end{pmatrix}$, $b_m = \rho$

Where, T_1, T_2 are smaller tridiagonal matrices.

D is a diagonal matrix containing the eigenvalues of the smaller tridiagonal matrices.

Q_1, Q_2 are eigenvector matrices after eigendecomposition of smaller tridiagonal

matrices.

$$u = \begin{pmatrix} Q_1^T & 0 \\ 0 & Q_2^T \end{pmatrix} v = \begin{pmatrix} \text{last column of } Q_1^T \\ \text{first column of } Q_2^T \end{pmatrix}$$

Step 2 Sorting using permutation matrix

Following equation is used for sorting process to arrange the eigenvalues and eigenvectors in an increasing order.

$$D + \rho uu^T = P^T P (D + \rho uu^T) P^T P \quad (9)$$

Where P is the permutation matrix.

Step 3 Reducing the computation using deflation

Deflation occurs when

1. ρ is very small, 2. smaller weights (u_i) due to smaller components in the vector used for rank one modification and 3. multiple eigenvalues [13]. When deflation occurs eigenvalues and eigenvectors do not require updation. Hence a saving in computation is achieved.

Step 4 Formation and solution of secular equation

$$|(D + \rho uu^T) - \lambda I| = 0$$

$$|(D - \lambda I)(I + \rho(D - \lambda I)^{-1} uu^T)| = 0$$

$$\text{Since, } |(D - \lambda I)| \neq 0, |(I + \rho(D - \lambda I)^{-1} uu^T)| = 0$$

Now,

$$|(I + \rho(D - \lambda I)^{-1} uu^T)| = 1 + \rho u^T (D - \lambda I)^{-1} u = 1 + \rho \sum_{i=1}^n \frac{u_i^2}{d_i - \lambda} \quad (10)$$

$$\begin{pmatrix} D_1 & 0 \\ 0 & D_2 \end{pmatrix} + b_m uu^T = [R][A][R^T] \text{ (obtained by solving secular equation)} \quad (11)$$

Eigenvalues and eigenvectors are obtained by solving a secular equation like equation (10) [13]-[14].

Step 5 Combining the solutions from Step IV and V the complete eigensystem of the symmetric tridiagonal matrix is obtained

$$T = [Q] \left(\begin{pmatrix} D_1 & 0 \\ 0 & D_2 \end{pmatrix} + b_m uu^T \right) [Q]^T = [Q][R][A][R]^T [Q]^T = [X][A][X]^T \quad (12)$$

Where, $[X] = [Q][R]$.

Hence,

$$A = [U][T][U]^T = [U][X][A][X]^T [U]^T = [V][A][V]^T \quad (13)$$

There are some issues related to divide and conquer algorithm. They are discussed below.

Sorting with Permutation Matrices [20] If $d_1 < d_2 \dots < d_n$ then the sequence of eigenvalues obtained will be $\lambda_1 < \lambda_2 < \dots < \lambda_n$. However, we may not come across a diagonal matrix with sorted diagonal elements after rank one modification and QR or QL iteration. Hence, we need to apply permutation to sort them in ascending order using permutation matrix.

$$\text{Let, } D = \begin{pmatrix} 13.1247 & 0 & 0 & 0 \\ 0 & 201.9311 & 0 & 0 \\ 0 & 0 & 0.0693 & 0 \\ 0 & 0 & 0 & 26.7189 \end{pmatrix}$$

$$\text{Let, } u = \begin{pmatrix} -0.5421 \\ -0.4540 \\ 0.2128 \\ -0.6743 \end{pmatrix}$$

Now applying permutation using permutation matrix, $P = \begin{pmatrix} 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 1 & 0 & 0 \end{pmatrix}$ and using

$$\text{equation (9) , we obtain } D(\text{sorted}) = \begin{pmatrix} 0.0693 & 0 & 0 & 0 \\ 0 & 13.1247 & 0 & 0 \\ 0 & 0 & 16.7189 & 0 \\ 0 & 0 & 0 & 201.9311 \end{pmatrix} \text{ and}$$

$$u(\text{modified}) = \begin{pmatrix} 0.2128 \\ -0.5421 \\ -0.6743 \\ -0.4540 \end{pmatrix}.$$

Solving Secular Equation The roots of the secular equation $1 + \rho \sum_{i=1}^n \frac{u_i^2}{d_i - \lambda}$ are the eigenvalues of the original matrix. ρ (rho) is the sub diagonal entry which creates the rank one modification. u_i^2 is the weight over the pole. The secular equation has poles at the eigenvalues of D and zeros at the eigenvalues of $D + \rho uu^T$.

According to interlacing property :

1. If rho is greater than zero, the roots lie in such a manner that : $d_1 < \lambda_1 < d_2 < \lambda_2 < \dots < d_n < \lambda_n$
2. If rho is less than zero then : $\lambda_1 < d_1 < \lambda_2 < d_2 < \dots < \lambda_n < d_n$

Assuming rho is greater than zero for $i < n$, the roots lie in between d_i and d_{i+1} , but for $i = n$, the root lies in a manner that $d_n < \lambda_n < d_n + \rho uu^T$ [13].

For the given matrix $\begin{pmatrix} 16.7118 & 10.7270 \\ 10.7270 & 34.2341 \end{pmatrix}$, where rho (10.7270) is greater than zero, the nature of the roots are examined below

The eigenvalues of the previously stated matrix are $\begin{pmatrix} 11.6228 & 0 \\ 0 & 39.3231 \end{pmatrix}$,

where the diagonal matrix D is $\begin{pmatrix} 5.9848 & 0 \\ 0 & 23.5071 \end{pmatrix}$. In Figure 1 we can see bold lines and vertical dotted lines. The points, where the bold lines cross the real axis at zero, are the roots for the corresponding matrix. The vertical dotted lines represent the diagonal elements after the rank one modification and eigendecomposition of smaller matrices.

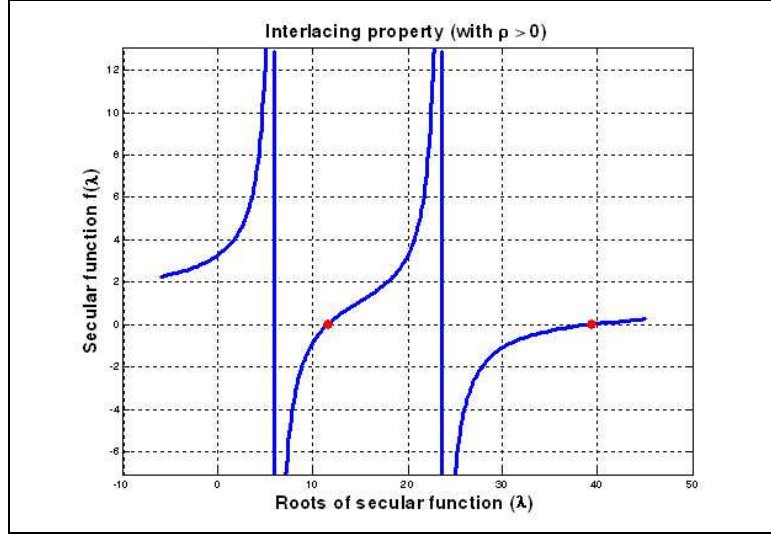


Fig. 1. Roots showing interlacing property with $\rho > 0$

For a matrix $\begin{pmatrix} 16.7118 & -10.7270 \\ -10.7270 & 34.2341 \end{pmatrix}$ with ρ (-10.7270) less than zero, the roots appear as shown in Figure 2.

The eigenvalues of the above said matrix with $\rho < 0$ are $\begin{pmatrix} 11.6228 & 0 \\ 0 & 39.3231 \end{pmatrix}$, where the diagonal matrix D is $\begin{pmatrix} 27.4388 & 0 \\ 0 & 44.9611 \end{pmatrix}$. The interlacing property is clearly found from Figures 1 and 2.

Methods of Solving Secular Equation [18] We assume that ρ (ρ) is greater than zero. The solution of secular equation is based on function approximation. So we may think of using Newton's iterative procedure to solve the equation, but Newton's method is based on local linear interpolation. At some points of initial guess (λ_0) for the desired root, the linear approximation would be horizontal and the next approximation would be a large negative number which is not useful [5]. This happens particularly when the weights are small. Therefore, we go for rational osculatory interpolation. Rational osculatory interpolation of secular function is the combination of two kinds of rational functions. If the secular function is $f(\lambda)$, the two rational functions are respectively $\Phi(\lambda)$ and $\Psi(\lambda)$. $\Phi(\lambda)$ is the sum of positive terms and $\Psi(\lambda)$ is the sum of negative terms. The inequality among them is $-\infty < \Psi(\lambda) < 0 < \Phi(\lambda) < +\infty$. In the following section, we will explore a set of schemes for solving secular equation. The first scheme is named as *approaching from left* because the algorithm will produce a sequence of monotonically increasing approximations to desired root provided

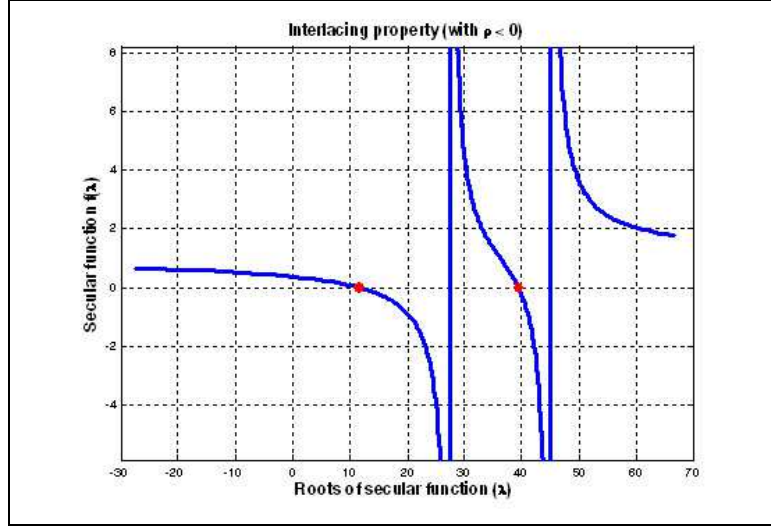


Fig. 2. Roots showing interlacing property with $\rho < 0$

the initial guess is less than the desired root. Starting with the initial guess which is less than the desired root, the scheme will yield a sequence of approximations converging monotonically upwards to the desired root. On the other hand, if the initial guess exceeds the required root so much that the next approximation does not converge, the scheme fails to get the desired root. *Approaching from left* scheme proves better for $i < n$, but for $i = n$ according to the interlacing property the root lies in the range $d_n < \lambda_n < d_n + \rho u u^T$. Here the initial guess exceeds the desired root and the second scheme i.e. *approaching from right* will yield a sequence of approximations converging monotonically downwards to desired root. A *middle way method* has been adopted by combining both the above schemes. This scheme takes both nearby poles into consideration, while the other two schemes take each of the nearby poles into consideration. Here the scheme handles two parts. The first part is, if the combination of two rational functions is greater than zero, the root is closer to d_i and if the function is less than zero, the root is closer to d_{i+1} . Interpolation of both $\Psi(\lambda)$ and $\Phi(\lambda)$ is carried out by taking both nearby poles into consideration. Following [5],

$$h(\lambda) = \frac{c_1}{d_i - \lambda} + \frac{c_2}{d_{i+1} - \lambda} + c_3 \quad (14)$$

$$\begin{aligned} f(\lambda) &= 1 + \sum_{k=1}^i \frac{u_k^2}{d_k - \lambda} + \sum_{k=i+1}^n \frac{u_k^2}{d_k - \lambda} \\ &= 1 + \Phi(\lambda) + \Psi(\lambda) \end{aligned} \quad (15)$$

Where coefficients c_1, c_2 and c_3 are computed by interpolating $f(\lambda)$. In [18] the above mentioned secular function has been represented similarly using the

system's matrix as:

$$D + \frac{1}{\rho}uu^T \quad (16)$$

$$h(\lambda) = \rho + r + R + \frac{s}{d_k - \lambda} + \frac{S}{d_{k+1} - \lambda} \quad (17)$$

Where $r + \frac{s}{d_k - \lambda}$ is approximated to $\Psi(\lambda)$ and $R + \frac{S}{d_{k+1} - \lambda}$ is approximated to $\Phi(\lambda)$. The initial guess (λ_0) plays a very significant role in finding solution for secular equation. For $k < n$ the initial guess is calculated as $\frac{d_k + d_{k+1}}{2}$ and for $k = n$ the initial guess is $d_n + \rho uu^T$. So with λ_0 , we calculate the value of $h(\lambda)$, and if the rational function is greater than zero, the root is closer to d_i and if it is less than zero, the root is closer to d_{i+1} . So according to the placement of the root, if it is close to d_i then we shift each of λ_0 , d_i and d_{i+1} by subtracting d_i from them. With the new value of the initial guess (λ_1), we solve the quadratic equation formed by the two rational functions, and η (iterative correction) is found. The desired root is found by computing $(\lambda_1 + \eta)$. We know that the weights over the poles play a major role in the solution of secular equation. One of the circumstances is associated with weights. In order to overcome this, fixed weight method was implemented with the structure same as middle way method. By combining *middle way method* and *fixed weight method*, a hybrid scheme was designed to make iteration faster. In this scheme, the function interpolation was carried out with d_k , d_{k+1} and d_{k-1} . The rational function for hybrid scheme is given below.

$$h(\lambda) = c + \frac{s}{d_{k-1} - \lambda} + \frac{u_k^2}{d_k - \lambda} + \frac{S}{d_{k+1} - \lambda} \quad (18)$$

Where $c = \rho + r + R$. Once the eigenvalues are obtained, eigenvectors can be computed using the following equation

$$x = \frac{(\lambda I - D)^{-1}u}{\|(\lambda I - D)^{-1}u\|} \quad (19)$$

Unfortunately, the above equation does not produce accurate eigenvectors [17,20] and the following equation is used for modifying u vectors where u vectors are updated and used in equation (19)

$$u_k = \sqrt{\frac{\prod_{j=1}^{k-1}(d_k - \lambda_j) \prod_{j=k}^n(\lambda_j - d_k)}{\rho \prod_{j=1}^{k-1}(d_k - d_j) \prod_{j=k+1}^n(d_j - d_k)}} \quad (20)$$

In this way, the eigenvectors computed (using equation 19) are relatively accurate [17].

Apart from the above SVD algorithms, *Bisection with Inverse Iteration* can also be used. This is specially used when singular values or eigenvalues are required within a specified range. This algorithm suffers from loss of orthogonality among the computed singular vectors or eigenvectors. Hence forced re-orthogonalization is necessary.

3 Case Study : Real-time Face and Eye Tracking

A fast SVD based face tracking scheme based on eigen space is shown in Figure 3. The scheme is composed of two sections - hardware and software. Fast and fixed-point SVD is required in this scheme for rapid customization of the system for a particular human subject. There are two paths: path 1 is for training and path 2 is for tracking purposes. The subject is asked to put its face within a box area (frame) in the image. The subject is then required to follow some instructions which direct him to make different poses with different face orientations. The training set is prepared online from extracted face images with different face orientations (front, front up, front down, looking left, looking right etc.) and it is customized for the person sitting in front of the camera. The covariance matrix is constructed from extracted face images with different poses at a frame rate of 30 fps. Fast SVD is performed on the covariance matrix to prepare the eigen space or feature space of face or eye in real-time. Now face and eye detection and tracking is carried out by projecting a block image from the incoming frame (from web camera) onto the feature vector space and comparing the reconstructed image with a standard face or eye image. Face and eyes are detected at approximately 10 fps in the laboratory environment.

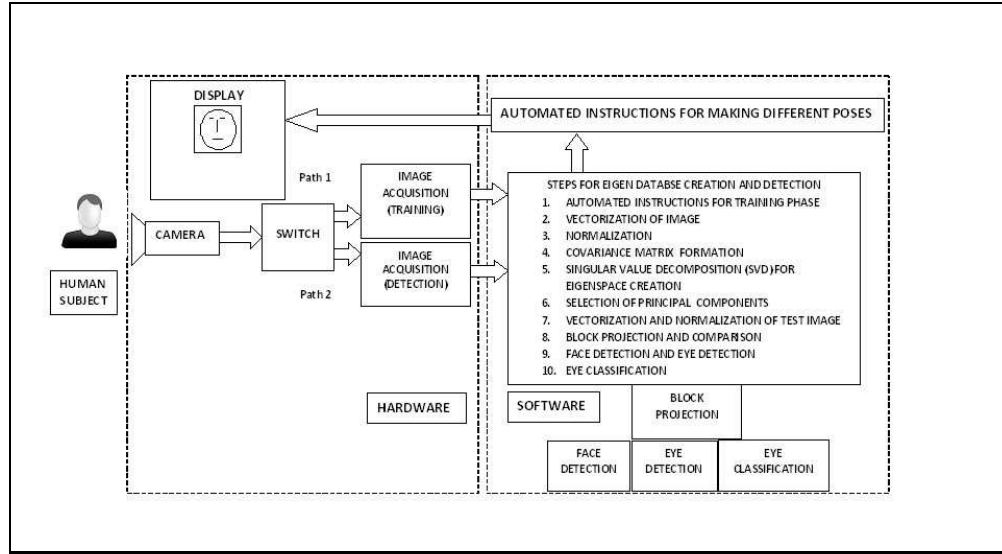


Fig. 3. Schematic diagram of the proposed online training based face and eye tracking system.

The steps for the scheme mentioned above are given below:

Eigenfaces are calculated as in [4]:

Step - 1: Obtain face images I_1, I_2, \dots, I_M (for training), each of dimension say

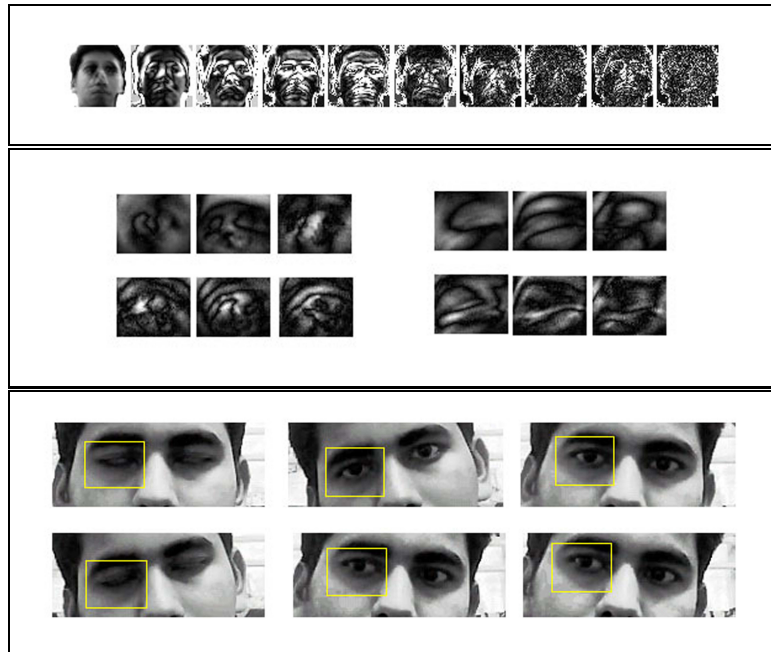


Fig. 4. EigenFaces (top), Eigen Eyes (for open and closed eyes) and Eye Detection using Fast SVD based algorithm (bottom).

$N \times N$.

Step - 2: Represent every image I_i as a vector Γ_i (of dimension $N^2 \times 1$).

Step - 3: Compute the average or mean face vector Ψ :

$$\Psi = \frac{1}{M} \sum_{i=1}^M \Gamma_i \quad (21)$$

Step - 4: Subtract the mean face from each face vector Γ_i :

$$\Phi_i = \Gamma_i - \Psi \quad (22)$$

Step - 5: The covariance matrix C given by:

$$C = \frac{1}{M} \sum_{n=1}^M \Phi_n \Phi_n^T = AA^T (N^2 \times N^2) \quad (23)$$

where $A = [\phi_1, \phi_2, \dots, \phi_M]$ ($N^2 \times M$ matrix) is very large, compute $A^T A$ instead ($M \times M$ matrix, $M \ll N$).

Now $AA^T = U \Sigma \Sigma^T U^T$ and $A^T A = V \Sigma^T \Sigma V^T$.

Step - 6: Compute the singular vectors v_i of $A^T A$. Using the equation $u_i = Av_i$ [14], singular vectors u_i of AA^T are obtained.

Step - 7: Depending on energy content of the components, keep only K singular vectors or eigenvectors corresponding to the K largest singular values. These K singular vectors are the eigenfaces corresponding to the set of M face images.

Step - 8: Normalize these K eigenvectors so that $u_i = \frac{u_i}{\|u_i\|_2}$.

Step - 9: Given a test image Γ , subtract the mean image Ψ : $\Phi = \Gamma - \Psi$.

Project the normalized test image onto the face space and obtain weight vector, $\Omega = U^T \Phi$, where $\Omega = [\omega_1 \omega_2 \dots \omega_k]$.

Step - 10: Compute reconstructed image $\hat{\Phi} = \sum_{i=1}^k w_i u_i$, ($w_i = u_i^T \Phi$).

Step-11: Compute error, $e = \|\Phi - \hat{\Phi}\|$.

Step-12: Classify test image as belonging to face class image for which the error norm e is minimum.

PERcentage eye CLOSure (PERCLOS) is defined as the proportion of time for which the eyelid remains closed more than 70-80% within a predefined time period. Level of drowsiness can be judged based on the PERCLOS threshold value [29]. Let N_a be the number of eye frames belong to the open or attentive category out of N_m number of eye frames captured in a minute. Hence $(N_m - N_a)$ is the number of eye frames belonging to the inattentive category. Then PERCLOS value per minute is

$$PERCLOS = \frac{N_m - N_a}{N_m} \times 100\% \quad (24)$$

However, the PERCLOS value computed per minute is not the correct measurement of drowsiness. Literature suggests that 20 minutes bout to bout measurement is a prominent indicator of drowsiness [29],[30]. So a trade-off between execution time and accuracy of PERCLOS is required. PERCLOS measured within

a 3 minute time interval is found to indicate the level of drowsiness reasonably well. A threshold over the PERCLOS value can be used to indicate level of drowsiness [30]. A comparison of eigenspace preparation using QR and QL and divide and conquer algorithm is shown in Table 3.

Table 6. Comparison of Execution Time (sec) of QL and QR and Divide and Conquer Algorithm

Order	QR and QL Algorithm	Divide and Conquer Algorithm
50	0.046875	0
100	0.046875	0.03125
200	0.15625	0.03125
500	1.671875	0.046875
1000	9.421875	0.09375

4 Conclusion and Future Scope

In this article different symmetric SVD algorithms have been evaluated for real-time preparation of eigen space for face and eye tracking to assess the level of alertness of human driver. Different symmetric SVD algorithms have been discussed with their complexity, advantages and disadvantages. Issues related to fast Divide and Conquer SVD algorithm have been discussed. A case study of real-time face and eye tracking has been illustrated with comparison of execution time for eigen space preparation with QR and QL and Divide and Conquer SVD algorithms. The scheme has been implemented in a workstation with Intel quad processor (2.5 GHz) and 3 GB DDR RAM.

The present work has been undertaken with an aim to implement the algorithm in an embedded platform where processing speed and other constraints exist. Implementation of fixed-point fast SVD for face and eye tracking could be useful for fixed-point DSPs and other fixed-point embedded platforms like ARM. To convert floating-point algorithms into fixed-point, it is necessary to first estimate the range of the required floating-point variables followed by choice of Q-format. Then the floating-point variables and floating-point arithmetic equations are converted to fixed-point format [22],[24]. The fixed-point algorithm is then tested for error and the algorithm is optimized for our purpose. This fast and fixed-point SVD algorithm could be used for other embedded signal and image processing applications. Development of other fixed-point signal processing algorithms like Wavelet Transform, Hidden Markov Model could be few future extension.

References

1. Yang M-H., Kriegman D. J. and Ahuja N.: Detecting Faces in Images: A Survey", IEEE Transactions on Pattern Analysis and Machine Intelligence, Vol. 24, No. 1,

- January 2001.
2. Hjelms E. and Low B. K.: Face Detection: A Survey, *Computer Vision and Image Understanding* 83, 236-274 (2001).
 3. Sirovich L. and Kirby M.: Low-dimensional Procedure for the Characterization of Human Faces, *Journal of the Optical Society of America A*, Vol. 4, P 519, March 1987.
 4. Turk M. and Pentland A.: Eigenfaces for Recognition, *Journal of Cognitive Neuroscience* Volume 3, Number 1, 1991.
 5. Demmel J. W.: *Applied Numerical Linear Algebra*, SIAM, Philadelphia, 1997.
 6. Golub G. H. and Van Loan C. F.: *Matrix Computations*, Third Editions, The Johns Hopkins University Press, Baltimore and London, 1996.
 7. Datta B. N.: *Numerical Linear Algebra and Applications*, Brooks/Cole Publishing Company, CA, USA, 1995.
 8. Parlett B. N.: *The Symmetric Eigenvalue Problem*, SIAM, Philadelphia, 1998.
 9. Hestenes M. R.: Inversion of Matrices by Biorthogonalization and Related Results", *SIAM*, Vol. 6, No. 1, March 1958, pp. 51-90.
 10. Svensson G.: A Block-Hestenes Method for the SVD, hem.passagen.se/-gohel/num/hest.ps, September 1989.
 11. Golub G. and Kahan W.: Calculating the Singular Values and Pseudo-Inverse of a Matrix, *J. SIAM Numer. Anal.*, Ser. B, Vol. 2, No. 2, 1965.
 12. Demmel J., Kahan W.: Accurate Singular Values of Bidiagonal Matrices, *SIAM J. Sci. Stat. Comput.*, v. 11, n. 5, pp. 873-912, 1990.
 13. Bunch J. R., Nielsen C. P. and Sorensen D. C.: Rank-One Modification of the Symmetric Eigenproblem, *Numer. Math.*, 31, 31-48 (1978).
 14. Cuppen J.J.M.: A Divide and Conquer Method for the Symmetric Tridiagonal Eigenproblem", *Numer. Math.* 36, 177-195 (1981).
 15. Zhou B. B. and Brent R. P.: On Parallel Implementation of the One-sided Jacobi Algorithm for Singular Value Decompositions", <http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=389182>.
 16. Stewart G. W.: Perturbation Theory for the Singular Value Decomposition", UMIACS-TR-90-124, CS-TR 2539, September 1990.
 17. Gu M. and Eisenstat S.C.: A Divide-and-Conquer Algorithm For The Symmetric Tridiagonal Eigen Problem, Research Report YALEU/DCS/RR-932, 27 November 1992.
 18. Li R. C.: Solving Secular Equations Stably and Efficiently, Computer Science Division Technical Report UCB//CSD-94-851, University of California, Berkeley, CA 94720, December, 1994.
 19. Melman A.: A numerical comparison of methods for solving secular equations, *Jour. of Comp. and App. Math.* 86(1997) 237-249.
 20. Rutter J.: A Serial Implementation of Cuppen's Divide and Conquer Algorithm for the Symmetric Eigenvalue Problem. <http://www.netlib.org/lapack/lawnspdf/lawn69.pdf>.
 21. Anderson E., Bai Z., Bischof C., Blackford S., Demmel J., Dongarra J., Du Croz J., Greenbaum A., Hammarling S., McKenney A., Sorensen D.: *LAPACK Users' Guide*, Third Edition, 22 Aug, 1999.
 22. Nikolić Z., Nguyen H. T., Frantz G.: Design and Implementation of Numerical Linear Algebra Algorithms on Fixed Point DSPs, *EURASIP Journal on Advances in Signal Processing*, Vol. 2007, Article ID 87046, doi: 10.1155/2007/87046.
 23. Cilio A. G. M., and Corporaal H.: Floating point to fixed point conversion of C code, *Proceedings of CC 99, the 8th International Conference on Compiler Construction (Lecture notes in computer science 1575)*, March 1999.

24. Kim S., Kum K., and Sung W.: Fixed-Point Optimization Utility for C and C++ Based Digital Signal Processing Programs, *IEEE Transaction on Circuits and Systems II: Analog and Digital Signal Processing*, pp. 1455–1464, vol. 45, no. 11, Nov 1998.
25. Keding H., Willems M., Coors M., and Meyr H.: FRIDGE: A fixed-point design and simulation environment, in *Proc. Design Automation and Test in Europe*, 1998, pp. 429–435.
26. Brent R. and Luk F.: The solution of singular value and symmetric eigenvalue problems on multiprocessor arrays, *SIAM J. Sci. Stat. Comput.*, 6, 1985, pp. 69–84.
27. Wang H., Leray P., Palicot J.: A CORDIC-based dynamically reconfigurable FPGA architecture for signal processing algorithms, *URSI 08, The XXIX General Assembly of the International Union of Radio Science*, Chicago IL, 2008.
28. Szecówka P. M. and Malinowski P.: CORDIC and SVD Implementation in Digital Hardware, *MIXDES 2010, 17th International Conference "Mixed Design of Integrated Circuits and Systems"*, june 24–26, 2010.
29. Dinges, D.F., Mallis, M., Maislin, G., Powell, J.W., 1998. Final Report: Evaluation of Techniques for Ocular Measurement as an Index of Fatigue and as the Basis for Alertness Management (Report No. DOT HS 808 762). National Highway Traffic Safety Administration, Washington, DC.
30. Dinges, D.F., Maislin, G., Brewster, R.M., Krueger, G.P., Carroll, R.J., 2004. Pilot Test of Fatigue Management Technologies: Final Report. Federal Motor Carrier Safety Administration, Washington, DC.